公共技术平滑落地的探索与实践

税友集团-体验技术团队-史泓

我给自己贴的标签

从业经历丰富

研发架构管理

前端搞了 七八年

从作坊到中大厂

2C2B2PPT

那些年搞过的公共技术

在 阿里 搞过 tarot

在挖财参与 wax, 搞过点石引擎

都失败了....

刚失败的我

都怪行业不景气，B2B 起不来，互金黄得快

一些年后

几经反思的我

让行业背锅是不对的

主因

我们一直只用一条腿走路才是导致公共技术落地如此艰难

公共技术落地的两条腿

- 理论探索 → 忽视或者从未用过的那条腿
- 技术手段 → 经常使用的甚至快用瘸了的那条腿

公共技术落地的困境

- 公共技术代表的主流程标准化和业务定制流程之间的矛盾
  - 带有公共属性的业务代码到底是公共组来维护，还是业务组来维护？
  - 现在只有 20% 的团队用，那将来呢？静态的看问题可能就是在挖坑.

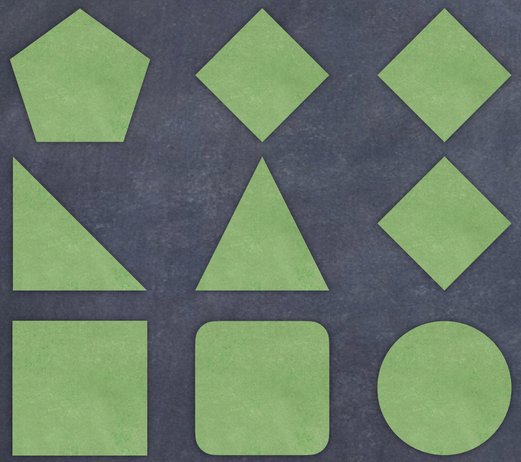
实施和推动公共技术落地的意义

降低前端工程的复杂度，让工程"形状"趋于一致

提升前端工程化水平

前端工程的复杂度

各种不同形状的工程



降低工程复杂度

公共技术消化降低复杂度

“复杂度不是静态的，是动态变化的”

公共技术的实施策略

- 锁定存量，确保实施的平滑性，静态的看待复杂度
- 控制增量，减少实施的成本，动态的看待复杂度

基于上述理论的技术实践

我们团队的前端工程复杂度

- 技术栈不统一，Backbone, Vue, React 并存
- 目录规范不一致，脚手架差异化，相同逻辑的不同实现
- 外部调用各有一套，不同的三方依赖解决相同的问题
- 工程数量多，覆盖业务广

乱!

“从客户端调用公共化切入，跑通最小可复制路径”

实施策略实践

- 锁定存量 → 直接从业务工程迁移代码到新的公共工程，只做路径修改
- 控制增量 → 客户端调用从对接到设计都交给对应的 owner 负责
  - 公共工程的 owner 负责 CodeReview
  - 其他团队成员可以持续贡献 Code，做到工程解耦，人不解耦，基于 Git 开发规范交流比过去更紧密

关于 CodeReview

- CodeReview 的价值在于提升而非限制

关于 CodeReview 在业务中的实践

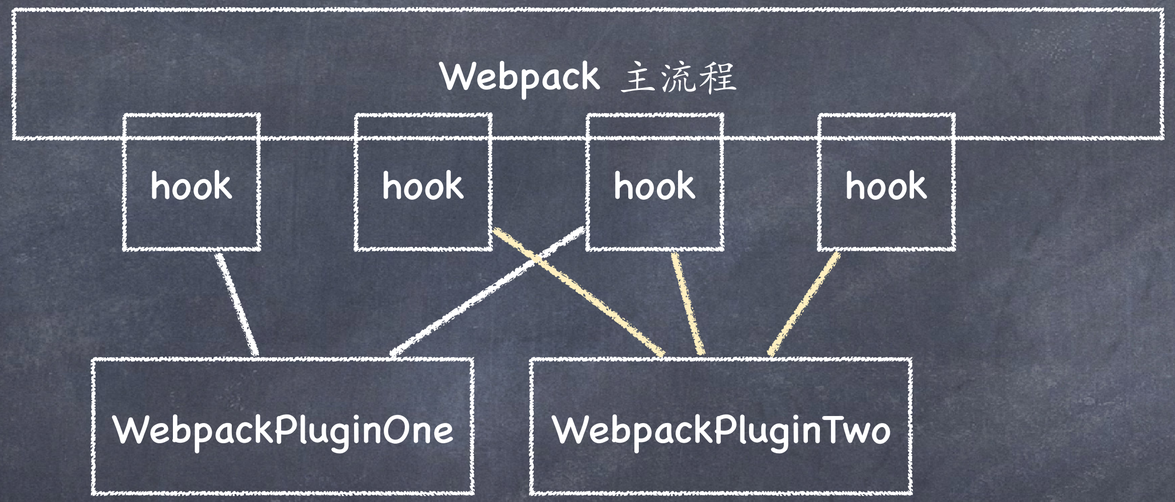
- 分组 Review
- 交叉统筹
- 按日 Review

公共化带来的问题

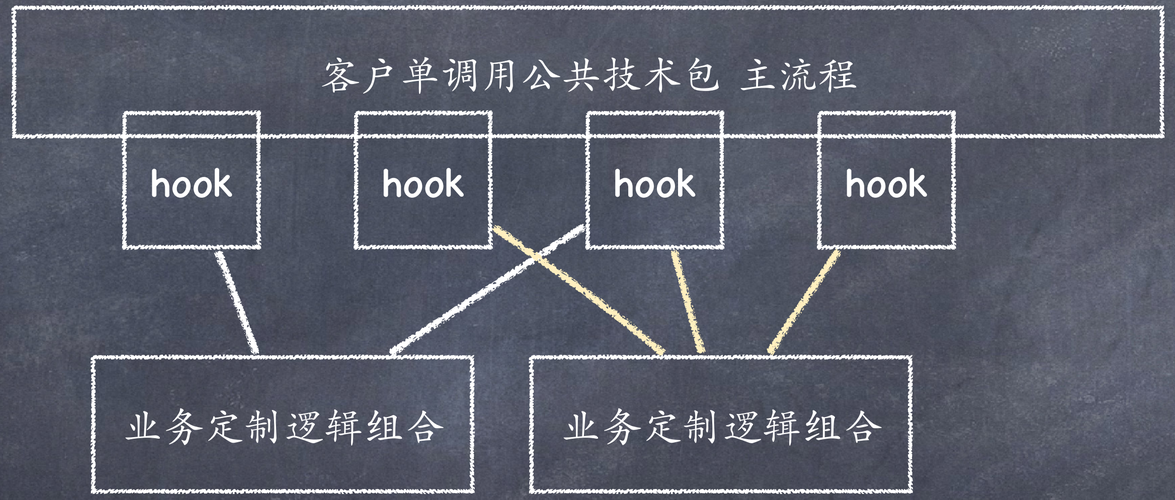
- 剥离出来的代码中既有属于公共的部分，又强依赖业务工程中的代码
  - 又回到了之前的问题
    - 带有公共属性的业务代码到底是公共组来维护，还是业务组来维护？

带来的灵感

Webpack 的 Plugin 系统



量身打造的公共技术插件系统



Hook 模式的核心依赖 → Tapable

Tapable 简介 https://juejin.im/post/6860688500392787982

伪代码演示...

Hook 模式和发布订阅的区别

“技术没有银弹，都是双刃剑”

使用 Hook 模式的注意事项

- 因为 hook 便捷好用，近乎 AOP 的体验，容易导致滥用，应该定期的审视某些 hook 的合理性
- 有些业务插件可能会变成公共插件，应该及时的消化在公共工程内，并移除相关定制的 hook
- 切忌!!! 不要在 main 逻辑内使用 hook 返回的数据，如果你有这种需求就要重新审视公共和业务代码的边界了，应该永远确保 hook 执行的逻辑对 main 是透明的，确保插件的可插拔性

谢谢，欢迎加微信交流